魂芯 DSP 的编译器设计与优化

王向前[1],洪 一[1,2],王 昊[2],郑启龙[3]
(1.合肥工业大学计算机与信息学院,安徽合肥 230009; 2.中国电子科技集团公司第三十八研究所,安徽合肥 230088;
3.中国科学技术大学计算机科学与技术学院,安徽合肥 230027)

摘 要: 魂芯 DSP 是一款字寻址的、分簇结构的、支持 SIMD 的 VLIW 处理器.介绍了基于开源编译器基础设施 open64 开发魂芯编译器的关键技术,包括地址寄存器的优化处理、综合多种启发因子的指令分簇、分簇架构下的寄存器分配和指令调度.介绍了魂芯 DSP 编译器的体系结构优化关键技术,包括基于依赖分析的向量化、高效指令的使用和零开销循环的识别.并总结开发经验,给出了基于开源编译基础设施开发编译器的若干注意点.

关键词: 地址寄存器; 分簇; 向量化; 零开销循环
中图分类号: TP314 **文献标识码:** A **文章编号:** 0372-2112 (2015)08-1656-06
电子学报 URL: http://www.ejournal.org.cn **DOI:** 10.3969/j.issn.0372-2112.2015.08.028

Compiler Design and Optimization for BWDSP

WANG Xiang-qian[1], HONG Yi[1,2], WANG Hao[2], ZHENG Qi-long[3]
(1. *School of Computer and Information, Hefei University of Technology, Hefei, Anhui* 230009, *China*;
2. *No.*38 *Research Institute, China Electronics Technology Group Corporation, Hefei, Anhui* 230088, *China*;
3. *School of Computer Science and Technology, University of Science and Technology of China, Hefei, Anhui* 230027, *China*)

Abstract: BWDSP is a word addressed VLIW DSP supporting clustering and SIMD. Based on open source compiling infrastructure open64, key technologies of compiler are developed for BWDSP which consist of optimized processing of address register, instruction clustering combined multi-heuristic factors, register allocation and instruction scheduling on clustering architecture. The key optimization technologies of BWDSP compiler on its hardware architecture include vectorization based on dependence analysis, application of effective instruction and recognization of zero overhead loop. Some general attention points for compiler development based on open source compiler infrastructure are presented after the development experience on BWDSP compiler is summarized.

Key words: address register; clustering; vectorization; zero overhead loop

1 引言

近年来,随着国家自主设计芯片能力的增强,涌现了一批成熟的产品级处理器,主要有中科院计算所研制的龙芯、上海高性能集成电路设计中心的申威、国防科大研制的银河飞腾、北大众志、苏州国芯、杭州中天等.为了突破市场,围绕自主芯片构建完善的"硬件平台-基础软件-应用"的生态系统成为产品研发的重要任务.其中,编译系统在该生态系统中占据至关重要的地位.

本文重点介绍魂芯 DSP 编译器的关键技术,主要包括基于开源编译基础设施开发编译器的后端移植技术,以及基于体系结构的优化技术.本文还提出了基于开源编译基础设施开发编译器的若干注意点,对于其它自主处理器平台编译器开发工作具有很好的指导意义.

2 魂芯 DSP 编译器设计

2.1 魂芯 DSP 体系结构

魂芯 DSP 是一款分簇结构、字寻址、支持 SIMD 的 16 发射的 VLIW 浮点运算信号处理器.片内有 3 块数据存储器,每块 8M bit.主要结构如图 1.魂芯 DSP 有 4 个计算簇,分别是 X 簇、Y 簇、Z 簇、T 簇,每个计算簇上有 8 个加法器,4 个乘法器,2 个移位器,每个计算核上有 64 个数据寄存器.计算簇与计算簇之间通过簇间传输总线通信.有 3 个地址簇即地址生成器,分别是 U 簇、V 簇、W 簇.存储器与计算核之间的数据交换所需的地址计算由地址生成器提供(AGU).AGU 是用作访存地址计算的特殊单元,每个 AGU 上有独立的地址寄存器文件(address register file),专用的地址运算器(address calcula-

收稿日期:2014-03-25;修回日期:2014-07-28;责任编辑:马兰英
基金项目:国家核高基重大专项(No.2012ZX01034001-001)

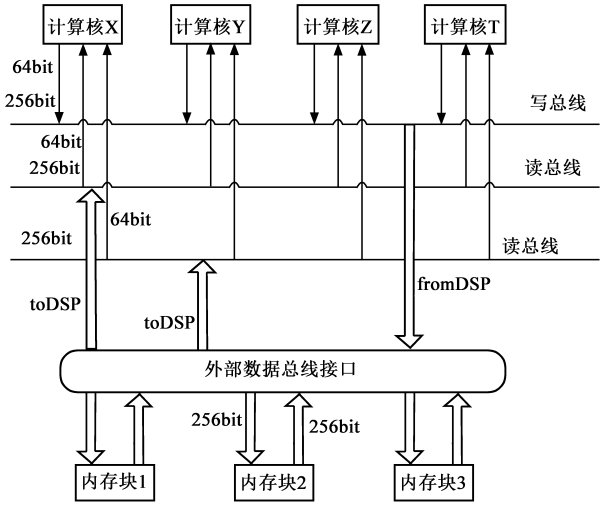

图1　魂芯DSP体系结构

tion ALU),内存存取单元(load/store unit).AGU 主要用于普通的地址加减计算,以及 load 和 store 指令.

2.2　open64 开源编译基础设施简介

open64[1~3]是一个 GPL 协议的工业级开源编译器,设计结构好,分析优化全面,是编译器高级研究的理想平台,被用在许多公司和大学的科研项目中.open64 前端是 GCC,后端则是建构在强大的 WHIRL 中间语言的基础上.它提供了良好的重定向机制,如图 2 所示.

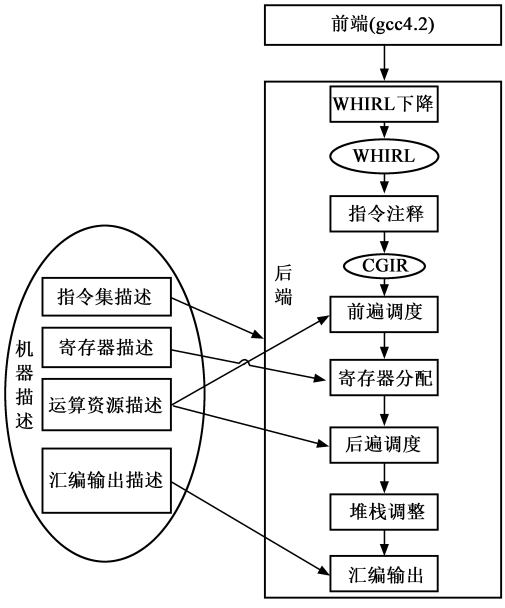

图2　open64重定向机制

3　open64 向魂芯 DSP 的重定向

3.1　魂芯 DSP 的字寻址模式支持

寻址问题是编译器设计过程中的基本问题.open64 前后端实现时假定针对的目标处理器架构是按照字节寻址的.为了支持魂芯 DSP 的字寻址模式,需要把 open64 前后端代码中涉及到字节寻址都统一修改为字寻址.open64 的前端 GCC 提供了良好的针对目标机器的抽象.可以修改 GCC 的机器描述文件 MD,把基本的寻址单元修改为 32bit,并以此达到 GCC 抽象语法树中所有数据类型的大小按照字来衡量的目的.后端则修改 WHIRL 结构所依赖的数据类型符号表,把数据类型的大小和对齐方式修改为按照字衡量时对应的数值.

3.2　魂芯 DSP 后端的实现

如图 2 所示,open64 向新的处理器的重定向主要工作包括 WHIRL 下降阶段的 ABI 传参的定制(包括实参和形参的定制)、机器描述、指令注释、指令分簇(为了支持魂芯 DSP 的分簇结构而增加)、寄存器分配、指令调度.在这些工作中,可以抽象出四个主要的问题,地址寄存器的优化处理、综合多种启发因子的指令分簇、分簇结构下的寄存器分配、分簇结构下的指令调度.

3.2.1　地址寄存器的优化处理

魂芯 DSP 的内存是分块的,有三个内存块 block0,block1,block2,如图 1 所示,可以支持分布在三个内存块上的两个读操作和一个写操作同时执行.此时的并行执行需要对应的访存地址的计算分布在三个地址生成单元.为了增加访存并行性的机会,魂芯 DSP 编译器上实现了分块内存分配算法如图 3 所示.它作为指令分簇的重要组成部分,解决了数据在分块内存的优化分布[4]和地址寄存器的分簇问题.分块内存分配算法主要分为三个步骤,如图 3 所示.

步骤 1　基于内存访问构建变量冲突图.如图 3 所示,算法依次遍历函数中 AGU 类型的指令(AGU 类型的指令主要包括访存指令、涉及到 AGU 的拷贝指令、AGU 单元上的其它运算指令等),根据 AGU 指令包含的变量和对应的地址寄存器构建变量列表;然后根据变量列表和对应的地址寄存器活跃变量范围是否有冲突构建变量冲突图.

步骤 2　分配变量到分块内存 block0、block1、block2.依据步骤一建立的变量冲突图,分配有冲突的变量到不同的内存块,这一点类似于图着色寄存器分配算法.需要注意的是,当没有可分配的内存块时,只能选择将两个有冲突的变量分配到同一个内存块上,这一点和图着色寄存器分配算法的内存溢出操作是不同的.

步骤 3　根据已经分配到分块内存的变量为相关地址寄存器分配 U/V/W 簇.首先依据已经分配内存块的变量为对应地址寄存器分配 U/V/W 簇信息,block0 对应的 AGU 簇信息为 U 簇,block1 对应的 AGU 信息为 V 簇,block2 对应的 AGU 信息为 W 簇;其次遍历整个函数为其它 AGU 类型指令分配 AGU 簇信息,此处分配的方法见 3.2.2 节.

```
Procedure AssignAGU(fun)fun:Function
begin
    interG: = BuildVarInterferencGraph(fun)
    AssignMemBlock(interG)
    AssignUVW(fun,interG)
End
Procedure BuildVarInterferencGraph(fun)
interG:InterGraph
fun:Function
begin
  for(each block b∈fun)
    for ceach instruction i∈b
    if(!(i∈AGUType))
        continue
    fi
    for(each operand o∈i.src)
      if (is_symbol(o))
        createVarNode(o,i.desc)
      fi
    od
  od
od
for(each Var v1∈VarNodeList)
  for(each Var v2∈VarNodeList)
    if (v1 = = v2)
        continue
    fi
    if(liveVange(v2.reg)⋂liveVange(v1.reg)! = ∅)

      CreateArc(v1,v2)
    fi
  od
od
return VarNodeList
End
```

图 3 分块内存分配算法

3.2.2 综合多种启发因子的指令分簇

魂芯 DSP 是分簇的体系架构,计算单元分 X/Y/Z/T 簇,地址单元也分 U/V/W 簇.为此,必须得设计一个高效的分簇算法.已经实现的基于寄存器压力的分簇算法[5],虽然解决了分簇问题,但未考虑寄存器传参、分块内存优化处理以及指令内在属性等启发性因素,从某种程度上它是一种随机算法.为了提高指令分簇的效率,在基于寄存器压力的分簇算法的基础上设计了综合多种启发因子的分簇算法,如图 4 所示.

启发因子(1) 传参寄存器.魂芯 DSP ABI 规定了通用寄存器和地址寄存器的传参规则.指令分簇之前,传参寄存器的簇信息是已知的.这些簇信息可以作为分簇算法的已知启发因子.

启发因子(2) 分块内存.3.2.1 节中已经分配到分块内存的变量为相关地址寄存器分配 U/V/W 簇.这些已经分配到分块内存上的变量对应的地址寄存器也可以作为分簇算法的已知启发因子.

启发因子(3) 指令的固有属性.魂芯 DSP 指令源操作数的簇信息是一样的.用 A 标识操作数是地址寄存器类型,用 C 标识操作数是通用寄存器类型,指令格式为目的操作数在前,源操作在后,则可以把指令分为以下几类:计算类指令(C - C)、地址类指令(A - A)、混合类型(要么是 C - A 类型,要么是 A - C 类型).例如计算类指令(C - C)是指该指令的寄存器操作数的簇信息要么全部在 X 簇、要么全部在 Y 簇、要么全部在 Z 簇、要么全部在 T 簇.而对于 A - C 类指令,则表示目的操作数的簇信息要么是 U 簇、要么是 V 簇、要么是 W 簇,而源操作数的簇要么全是 X 簇,要么全是 Y 簇,要么全是 Z 簇,要么全是 Z 簇.该启发因子是分簇算法的迭代启发因子,如图 4 所示.

```
Procedure Cluster(fun)
fun:Function
Begin
reglist:RegList
change:boolean
operlist:OperList
AssignAGU(func)
AssignCLusterABI(fun)
Reglist: = BuildRegList();
change: = true;
While(change)
  chang: = false
  for (each Reglist reg ∈ reglist)
    if(!(reg ∈ GREG))
      continue
    fi
    operlist: = getLiveUse(reg)
    for(each Oper op ∈ operlist)
      if(!(Is_Same_Cluster_CU_Op(op))
      continue
      fi
      for(each Operand operand ∈ op -> get_CU_operands())
      if(operand ! = reg && operand -> getC1user() = = Undefined)
          chang: = true
          AssignSameC1uster(reg,operand)
        fi
      od
    od
  od
od
RegPressurec1uster(fun)
end
```

图 4 综合多个启发因子的分簇算法

3.2.3 分簇结构下的寄存器分配

open64 的寄存器分配基于经典的图着色分配算法,

并提供良好的可移植接口.通过扩展机器描述中的寄存器文件描述即可以支持分簇框架下的寄存器分配.

open64 寄存器分配框架已有的整型寄存器类型、浮点寄存器类型不能满足魂芯 DSP 分簇结构的寄存器分配需要.在经过分簇阶段后,寄存器类型从通用寄存器(Integer)和地址寄存器(Address)转化为 IntegerX, IntegerY, IntegerZ, IntegerT, AddressU, AddresV, AddressW.按照机器描述规范描述每个簇对应的寄存器文件的可分配寄存器、caller 寄存器、callee 寄存器等信息等.在确保寄存器分配文件的机器描述正确的前提下,寄存器分配框架可高效正确地进行寄存器分配.

3.2.4 分簇架构下的指令调度

open64 的指令调度机制是由机器描述支持的微观资源调度和高级依赖图调度[6~8]组成,如图 5 所示.高级依赖图调度主要是根据指令的依赖关系(指令延迟信息),建立依赖图进行调度.微观资源调度是在构建指令各种资源限制的基础上,和高级依赖图进行交互,使得指令调度的结果既不违反指令依赖关系,又能满足硬件资源的需求.

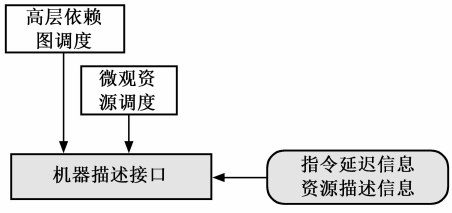

图5　调度框架

要使得该调度框架支持分簇结构,而分簇本身是一种硬件资源的分布组合.首先需要在机器描述中描述各个簇的运算资源、簇间传输通道、立即数等资源信息.有些资源是全局性的,在描述时可以归为某一簇.其次,魂芯 DSP 的一些特性,例如双字指令只能占用调度行的前几个指令槽、一个调度行中只能包括一个跳转指令等太过细节的限制,机器描述则无法进行描述,需要对微观资源调度器进行扩展,使得调度时把这些限制条件考虑在内.

4 体系结构优化

在实现 open64 向魂芯 DSP 的重定向,进行高强度的、专业的测试集测试,修正了大量错误后,一个支持魂芯 DSP 的基本编译器就得以产生.然而,为了提高编译器生成代码的效率,针对魂芯 DSP 体系结构进行优化,是必须的.魂芯 DSP 的体系结构优化主要包括向量化、高效指令的使用、零开销循环的识别等.

4.1 向量化

魂芯 DSP 体系结构的主要特性是向量化,包括访存向量化和计算向量化.为了充分发挥魂芯 DSP 的性能,编译器优化必须做好向量化这个环节.魂芯 DSP 编译器针对循环采取基于依赖分析的向量化算法[9~11],如图 6 所示;对未成功向量化的循环基本块和其它基本块采取超字级并行性技术(SLP)[12,13]向量化算法.

```
Procedure LoopVector(Loop, k)
loop:LOOP, k:int
Begin
  dependenceGraph: DependenceGraph
  dependenceGraph: = DepenceAnalysis(loop)
  for(each DependenceGraph g∈dependence Graph)
    if (g -> cycleflag = = true)
      if(k = = Loop -> innerest)then
        TryBreakCycle(g, loop)
      else
        begin
          InterChangeLoop(loop)
          GenerateLoopWraper(g)
          LoopVector(loop -> next, k + 1)
        end
      fi
    else
      GenerateVectoLoop(g)
    fi
  od
end
```

图 6　循环向量化算法

依赖分析是向量化的基础.循环的依赖可以分为循环携带依赖和循环无关依赖.循环中不携带依赖的任何单语句可以直接向量化.但这个只是向量化的充分条件.魂芯 DSP 循环向量化算法采取的是多维度向量化的递归算法.首先尝试在最外层循环生成向量代码.如果最外层循环是有环的依赖,妨碍向量化,则尝试用循环交换技术选择一个循环进行串行化之后,接着尝试更深一层的向量化,并忽略外层携带的依赖.如果是最内层循环并且存在有环依赖,尝试用标量扩展技术打破有环依赖之后再尝试向量化.如果某层循环不存在有环的依赖,就可以根据魂芯 DSP 的向量化指令生成向量化的循环.

4.2 高效指令的使用

魂芯 DSP 提供乘累加指令、求绝对值指令、求最大值指令、求最小值指令等加速指令,这些指令可以大幅提高相关计算的效率.open64 本身提供了识别这些指令的框架,只需要在指令注释阶段把这些指令用上即可.

4.3 零开销循环的识别

零开销循环[14,15]是 DSP 处理器中常见的体系结构特性.它可以加快循环执行的效率而不会引起生成代码的增大.典型 DSP 算法很大部分的执行时间花费在循环执行上.在循环正规式的基础上进行零开销的转

换,需要满足(1):循环体里不能含有函数调用;(2):转换为零开销循环的循环个数不能超过硬件提供的零开销循环个数;(3):循环规约变量必须是循环退出条件指令的操作数;(4):如果循环有多个出口,则该循环不能进行零开销循环转换;(5):如果循环次数不可数,则该循换不能进行零开销循环转换.

5 魂芯 DSP 编译器的性能测试

魂芯 DSP 体系结构和 ADI TS201 都是分簇结构、支持 SIMD 的、多发射的、分块内存的、支持 VLIW 的信号处理器,如表 1 所示,可见魂芯 DSP 整体并行能力大致是 TS201 的 4 倍左右.可以选择 ADI TS201 的 C 编译系统作为魂芯 DSP 编译器优化性能对比的参考,在程序有充足并行性的条件下,只要魂芯 DSP 的 C 编译器性能是 TS201 编译器性能的 4 倍左右,就说明魂芯 DSP 编译器的性能达到了比较高的水平,优化效率可匹敌 ADI TS201 的编译器.

表 1 TS201 和魂芯 DSP 体系结构特征对比

特性	TS201	魂芯 DSP
向量化	支持	支持
分簇结构	是	是
地址簇	2 个	3 个
计算簇	2 个	4 个
ALU 运算单元	2 个	8 个
乘法器	2 个	4 个
分块内存	是	是
发射宽度	4	16
面向领域	信号处理和图像应用	信号处理和图像应用

选取 DSP 经典的算法作为性能测试集(如表 2 所示),TS201 和魂芯 DSP 编译器的优化性能对比如图 7 所示.数据并行性高的程序如 FFT_radix4 和 VECTOR_SUM,魂芯 DSP 编译器的性能是 TS201 编译器性能 3~4 倍之多.而像 IIR 等实例,其本身的数据并行性不大,魂芯 DSP 编译器的编译性能反而不如 TS201 的编译器性能.至此,基于 open64 开发了一款针对魂芯 DSP 的优化编译器,其优化性能达到比较可观的水平.

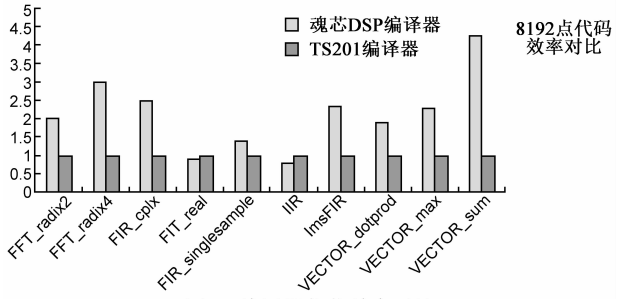

图7 编译器优化效率对比

表 2 基准测试集

测试基准	算法
FFT_radix2	基 2 快速傅立叶变换
FFT_radix4	基 4 快速傅立叶变换
FIR_cplx	复数有限冲激响应滤波器
FIR_real	实数有限冲激响应滤波器
FIR_singlesample	单样本有限冲激响应滤波器
IIR	无限冲激响应滤波器
lmsFIR	最小均方有限冲激响应滤波器
VECTOR_dotprod	向量点积
VECTOR_max	向量最大值
VECTOR_sum	向量求和

6 开源编译设施开发编译器的注意点

为了给他人基于开源编译基础设施开发编译器提供借鉴,特总结以下注意点.慎重选型.主流开源编译基础设施主要包括 GCC、open64、LLVM 等.要综合对比开源编译基础设施的主要特性,例如模块化、可重定向性、功能全集等等.考察开源编译基础设施与目标处理器体系结构的匹配度.选取一款与目标处理器架构的主要体系结构特点最匹配的开源编译基础设施,可以节省不少开发时间.着重结合目标体系结构的核心结构特征,重构或改造开源编译基础设施,并在此基础上开发体系结构优化算法.例如魂芯 DSP 的核心特征是分簇和向量化,高效率支持分簇结构和进行向量化优化就成为编译器开发的难点和重点.

7 结论

魂芯 DSP 是 VLIW 的 SIMD 体系架构的高性能信号处理器,它的指令级并行完全靠编译器.基于开源编译基础设施开发面向魂芯 DSP 的优化编译器,主要工作包括后端移植技术和体系结构优化技术.后端移植技术主要的关键技术是地址寄存器的优化处理、综合多种启发因子的分簇算法、分簇结构下的寄存器分配和指令调度.而体系结构优化的关键技术是向量化、高效指令的使用和零开销循环的识别等.魂芯 DSP 的优化编译器的健壮性和性能达到工业应用水平,为魂芯 DSP 的产业化奠定了坚实的基础.

参考文献

[1] Lin M, et al. Retargeting the open64 compiler to powerpcprocessor[A]. IEEE International Conference on Embedded Software and Systems Symposia, 2008[C]. Washington, DC: IEEE Com-

puter Society,2008.152 – 157.
[2] Malhotra V. Open64 compiler[A/OL]. http://www – vlsi.stanford. edu/smart memories/protected/ meetings/summer2003/Open64Compiler.pdf,2003.
[3] De S K,Dasgupta A,Kushwaha S,et al. Development of an efficientDSP compiler based on open64[A]. HSU Wei-Chung. Open64 Workshop at 2008 ACM International Symposium on Code Generation and Optimization[C]. New York:Association for Computing Machinery,2008.1 – 11.
[4] 郑启龙,等. DSP 分块内存和多 AGU 的编译指示优化[J].小型微型计算机系统,2012,33(003):582 – 586.
Zheng Qilong, et al. Compiler optimizations via progma for DSP local memory and multi-AGUs[J]. Journal of Chinese Computer Systems,2012,33(003):582 – 586.(in Chinese)
[5] 雷一鸣,等.一种基于寄存器压力的 VLIW DSP 分簇算法[J].计算机应用,2010,30(1):274 – 276.
Lei Yiming, et al. Register based algorithm for VLIW DSP cluster assignment[J]. Journal of Computer Applications,2010,30(1):274 – 276.(in Chinese)
[6] Ju R,Chan S,Chow F,et al. Open Research Compiler (ORC):Beyond Version 1.0[A/OL]. http://ipf-orc.sourceforge.net/ORC-PACT02-tutorial.pdf.2002.
[7] Wu C,Lian R,Zhang J,et al. An overview of the open research compiler[A]. 17th International Languages and Compilers for High Performance Computing Workshop, LCPC 2004[C]. Berlin Heidelberg:Springer,2005.17 – 31.
[8] Lin Y C,Tang C L,Wu C J,et al. Compiler supports and optimizations for PAC VLIW DSP processors[A]. 18th International Languages and Compilers for High Performance Computing Workshop,LCPC 2005[C]. Berlin Heidelberg:Springer,2006.466 – 474.
[9] Nuzman D,Rosen I,Zaks A. Autovectorization of interleaved data for SIMD[A]. Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation[C]. New York:Association for Computing Machinery,2006.132 – 143.
[10] Allen R,Kennedy K. Optimizing Compilers for Modern Architectures[M]. San Francisco:Morgan Kaufmann,2002.
[11] 李玉祥.面向非多媒体程序的 SIMD 向量化方法及优化技术研究[D].合肥:中国科学技术大学,2008.
Li Yuxiang. Research and optimization of SIMD Vectorization Algorithms on Non-multime dia Applications[D]. Hefei:University of Science and Technology of China,2008.(in Chinese)
[12] Larsen S, Amarasinghe S. Exploiting superword level parallelism with multi media instruction sets[A]. Proceedings of the ACM SIGPLAN 2000 conference on Programming language design and implementation[C]. New York:Association for Computing Machinery,2000.145 – 156.
[13] 魏帅,赵荣彩,姚远.面向 SLP 的多重循环向量化[J].软件学报,2012,23(7):1717 – 1728.
Wei Shuai,Zhao Rongcai,Yao Yuan. Loop-nest Auto-vectorization based on SLP[J]. Journal of Software,2012,23(7):1717 – 1728.(in Chinese)
[14] Uh G R,Wang Y,Whalley D,et al. Effective exploitation of a zero overhead loop buffer[A]. Proceedings of the ACM SIGPLAN 1999 workshop on Languages, compilers, and tools for embedded systems[C]. New York:Association for Computing Machinery,1999.10 – 19.
[15] Uh G R, et al. Techniques for effectively exploiting a zero overhead loop buffer[A]. 1999's Compiler Construction[C]. Berlin Heidelberg:Springer,2000.157 – 172.

作者简介

王向前　男,1985 年生,河南南阳人,合肥工业大学博士研究生.从事的主要研究方向为编译优化与系统软件优化.
E-mail:forward@mail.ustc.edu.cn

洪　一　男,1963 年生,安徽铜陵人,合肥工业大学教授、博士生导师、中国电科首席专家、"魂芯"DSP 总设计师.主要研究方向为信号处理与信号处理器体系结构设计.

王　昊　男,1977 年生,吉林松原人,中国电科第三十八所工程师.从事的主要研究工作为编译设计与优化.

郑启龙　男,1969 年生,安徽合肥人,中国科学技术大学副教授.主要研究方向为并行计算与高效能软件工具与环境.